View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by PORTO@iris (Publications Open Repository TOrino - Politecnico di Torino)

POLITECNICO DI TORINO
Repository ISTITUZIONALE

High-performance Implementations and Large-scale Validation of the Link-wise Artificial Compressibility Method

Original
High-performance Implementations and Large-scale Validation of the Link-wise Artificial Compressibility Method / Christian Obrecht; Pietro Asinari; Frédéric Kuznik; Jean-Jacques Roux. - In: JOURNAL OF COMPUTATIONAL PHYSICS. - ISSN 0021-9991. - STAMPA. - 275(2014), pp. 143-153.

Availability:
This version is available at: 11583/2550952 since:

Publisher:
Elsevier BV: PO Box 211, 1000 AE Amsterdam Netherlands: 011 31 20 4853757, 4853642 011 31 20, 011

Published
DOI:10.1016/j.jcp.2014.06.062

Terms of use:
openAccess

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

04 August 2020

High-performance Implementations and Large-scale Validation of the Link-wise Artificial Compressibility Method

Christian Obrecht[a,*], Pietro Asinari[b], Frédéric Kuznik[a], Jean-Jacques Roux[a]

[a] *CETHIL UMR 5008 (CNRS, INSA-Lyon, UCB-Lyon 1), Université de Lyon, France*
[b] *SMaLL, Dipartimento Energia, Politecnico di Torino, Italy*

Abstract

The link-wise artificial compressibility method (LW-ACM) is a recent formulation of the artificial compressibility method for solving the incompressible Navier–Stokes equations. Two implementations of the LW-ACM in three dimensions on CUDA enabled GPUs are described. The first one is a modified version of a state-of-the-art CUDA implementation of the lattice Boltzmann method (LBM), showing that an existing GPU LBM solver might easily be adapted to LW-ACM. The second one follows a novel approach, which leads to a performance increase of up to 1.8× compared to the LBM implementation considered here, while reducing the memory requirements by a factor of 5.25. Large-scale simulations of the lid-driven cubic cavity at Reynolds number $\mathrm{Re} = 2000$ were performed for both LW-ACM and LBM. Comparison of the simulation results against spectral elements reference data shows that LW-ACM performs almost as well as multiple-relaxation-time LBM in terms of accuracy.

Keywords: Computational fluid dynamics, Link-wise artificial compressibility method, High-performance computing, Lid-driven cubic cavity, CUDA

1. Introduction

Although the use of unstructured meshes is widespread in computational fluids dynamics (CFD), alternative approaches using Cartesian grids such as the lattice Boltzmann method (LBM) have gained increasing interest in recent years. However, while unstructured meshes are specifically intended for representing complex boundaries, Cartesian grid approaches need additional techniques to address this issue. Using nested meshes with hierarchical data structures such as octrees [8] is a possible method at the expense of the regularity of the data access pattern. Another way consists in incorporating additional treatments for boundary nodes based on techniques such as immersed boundary methods [11] or cut cell methods [10], inaccurate resolution of the boundary layers being a possible shortcoming. Nevertheless, from a computational standpoint, CFD solvers based on uniform Cartesian meshes are especially well-suited for high-performance implementations on massively parallel processors such as graphics processing units (GPUs) [16].

*To whom correspondence should be addressed. E-mail: christian.obrecht@insa-lyon.fr.

Sharing many similarities with the LBM, the artificial compressibility method (ACM) has been recently given a novel formulation, known as the link-wise ACM (LW-ACM) [3], which involves a finite set of links on a regular Cartesian mesh. Besides other interesting features, the LW-ACM enables to use specific techniques from both LBM and finite differences. In this paper, we describe two GPU implementations of the LW-ACM in three dimensions within the framework of the NVIDIA CUDA technology. Given the algorithmic similarities between LBM and LW-ACM, our first approach reinvests common GPU implementation techniques of the LBM [13]. However, the LW-ACM updating rule makes possible to recover all the necessary informations from the hydrodynamic variables of the fluid. Hence our second implementation takes advantage of this specific feature to reduce considerably the memory requirements as well as the amount of data transferred between GPU and device memory. In addition, for validation and comparison purposes, we performed large-scale simulations of the lid-driven cubic cavity using either LW-ACM or LBM, and matched our simulation results against highly accurate reference data.

The remainder of the paper is organized as follows. In Section 2, we briefly introduce the LW-ACM and discuss its algorithmic aspects. Section 3 describes the two GPU implementations of LW-ACM. In section 4, we report performance of both LW-ACM implementations and compare these results with a state-of-the-art CUDA LBM solver. Section 5 presents our simulations of the lid-driven cubic cavity and section 6 provides some concluding remarks.

2. Link-wise artificial compressibility method

2.1. Artificial compressibility equations

The artificial compressibility method (ACM), which was first introduced by Chorin in 1967 [6], is a numerical approach for solving the incompressible Navier–Stokes equations (INSE). Using the Einstein summation convention, the INSE are expressed as:

$$\partial_t u_i + u_j \partial_j u_i = -\frac{1}{\rho_0}\partial_i p + \nu \partial^2_{jj} u_i + F_i \,, \tag{1}$$

$$\partial_j u_j = 0, \tag{2}$$

where u_i are the components of the fluid velocity $\boldsymbol{u}$, p is the pressure, F_i are the components of the external force per unit mass, ρ_0 is the density, ν is the kinematic viscosity, and t is the time. The indices $i,\ j = 1,\ 2,\ 3$ refer to the spatial coordinates.

Since pressure does not appear in Eq. 2, i.e. the continuity equation, most mainstream numerical methods resort to the derived pressure Poisson equation:

$$\partial^2_{jj} p = -\rho_0 \partial^2_{ij} (u_i u_j). \tag{3}$$

It should be noted that the adoption of an implicit time-marching, although natural, impairs the adaptability of these approaches to massive parallelism.

The ACM is in contrast based on the artificial compressibility equations (ACE), a modified form of the INSE in which the continuity equation is replaced by:

$$\partial_t \rho + \partial_j u_j = 0, \quad p = \rho/\zeta, \tag{4}$$

where ρ is defined as the artificial density, ζ as the artificial compressibility, and $p = \rho/\zeta$ as the artificial equation of state. The ACE yield an artificial speed of sound: $c_s = 1/\sqrt{\zeta}$, and thus an artificial Mach number: $\mathrm{Ma} = \sqrt{\zeta} \times \max \|\boldsymbol{u}\|$.

The presence of the pressure time derivative in Eq. 4 allows for explicit time-integration. Although ACM was primarily intended for steady flows, it is known to yield also accurate solutions for the time-dependent INSE in the limit of vanishing Mach number [15].

2.2. Link-wise formulation

The LW-ACM is a discrete formulation of the ACM within a framework similar to the one of the LBM. It operates on a regular Cartesian spatial mesh of mesh size δx with a regular time step δt. In accordance with the established practice of LBM, we shall express all following quantities in terms of lattice units, i.e. adopt δx as unit of length and δt as unit of time.

The mesh is associated to a lattice stencil, i.e. a finite set of velocities $\{\boldsymbol{\xi}_\alpha\}$ where $\alpha = 0, \ldots, Q-1$. This stencil is usually chosen such as to link the mesh points to some of their nearest neighbours on the mesh. In the present work, we used the three dimensional D3Q19 stencil, which is represented in Fig. 1. The coordinates of the $\boldsymbol{\xi}_\alpha$ velocities in the D3Q19 stencil are defined as:

$$[\boldsymbol{\xi}_\alpha] = \begin{cases} (0,0,0) & \alpha = 0, \\ (\pm 1,0,0),(0,\pm 1,0),(0,0,\pm 1) & \alpha = 1, \ldots, 6, \\ (\pm 1,\pm 1,0),(\pm 1,0,\pm 1),(0,\pm 1,\pm 1) & \alpha = 7, \ldots, 18. \end{cases} \tag{5}$$

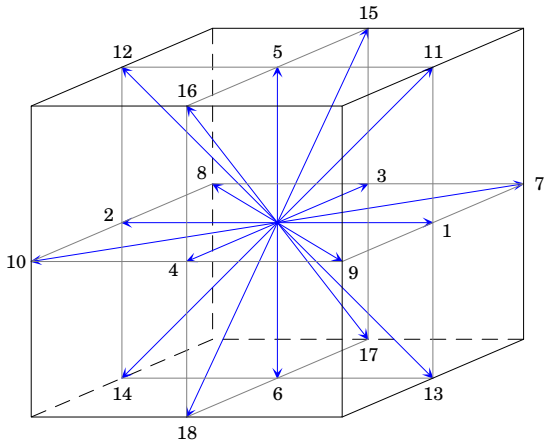


Figure 1: The D3Q19 stencil — *The blue arrows represent the $\boldsymbol{\xi}_\alpha$ velocities.*

The hydrodynamics is represented by a set of Q dependent variables $\{f_\alpha\}$ defined at the mesh points such that:

$$\rho = \sum_\alpha f_\alpha, \tag{6}$$

$$\rho \boldsymbol{u} = \sum_{\alpha} f_{\alpha} \boldsymbol{\xi}_{\alpha} . \tag{7}$$

In addition, we use local equilibrium functions $f_{\alpha}^{(\mathrm{e})}$ which are known functions of ρ and $\boldsymbol{u}$. With $c_{\mathrm{s}} = 1/\sqrt{3}$, the equilibria are written as:

$$f_{\alpha}^{(\mathrm{e})}(\rho, \boldsymbol{u}) = w_{\alpha} \rho \left(1 + 3\boldsymbol{u} \cdot \boldsymbol{\xi}_{\alpha} + \frac{9}{2} (\boldsymbol{u} \cdot \boldsymbol{\xi}_{\alpha})^2 - \frac{3}{2} \boldsymbol{u}^2 \right), \tag{8}$$

where w_{α} are the weights associated to the velocity set (see Appendix A of [3]). For the D3Q19 stencil, these weights are:

$$w_{\alpha} = \begin{cases} 1/3 & \alpha = 0, \\ 1/18 & \alpha = 1, \ldots, 6, \\ 1/36 & \alpha = 7, \ldots, 18. \end{cases} \tag{9}$$

Moreover, we denote $f_{\alpha}^{(\mathrm{e,o})}$ the odd parts of the equilibrium functions:

$$f_{\alpha}^{(\mathrm{e,o})}(\rho, \boldsymbol{u}) = \frac{1}{2} \left(f_{\alpha}^{(\mathrm{e})}(\rho, \boldsymbol{u}) - f_{\alpha}^{(\mathrm{e})}(\rho, -\boldsymbol{u}) \right). \tag{10}$$

Derived from the Bhatnagar–Gross–Krook model equation [4], the fundamental updating rule of the LW-ACM is expressed as:

$$f_{\alpha}(\boldsymbol{x}, t+1) = f_{\alpha}^{(\mathrm{e})}(\boldsymbol{x} - \boldsymbol{\xi}_{\alpha}, t) + 2 \left(\frac{\omega - 1}{\omega} \right) \left(f_{\alpha}^{(\mathrm{e,o})}(\boldsymbol{x}, t) - f_{\alpha}^{(\mathrm{e,o})}(\boldsymbol{x} - \boldsymbol{\xi}_{\alpha}, t) \right), \tag{11}$$

where ω is the relaxation frequency. Assuming diffusive scaling, i.e. $\delta x = \varepsilon$ and $\delta t = \varepsilon^2$ with $\varepsilon \ll 1$, asymptotic analysis shows that the scaled moments:

$$\bar{\rho} = (\rho - \rho_0)/\varepsilon^2, \tag{12}$$

$$\bar{\boldsymbol{u}} = \boldsymbol{u}/\varepsilon, \tag{13}$$

satisfy the ACE (see Appendix B of [3]), with viscosity:

$$\nu = \frac{1}{3} \left(\frac{1}{\omega} - \frac{1}{2} \right). \tag{14}$$

2.3. Algorithmic aspect

The updating rule of LW-ACM may be split into two steps:

$$f_{\alpha}(\boldsymbol{x}, t+1) = f_{\alpha}^{*}(\boldsymbol{x} - \boldsymbol{\xi}_{\alpha}, t) + 2 \left(\frac{\omega - 1}{\omega} \right) f_{\alpha}^{(\mathrm{e,o})}(\boldsymbol{x}, t), \tag{15}$$

$$f_{\alpha}^{*}(\boldsymbol{x}, t+1) = f_{\alpha}^{(\mathrm{e})}(\boldsymbol{x}, t+1) - 2 \left(\frac{\omega - 1}{\omega} \right) f_{\alpha}^{(\mathrm{e,o})}(\boldsymbol{x}, t+1), \tag{16}$$

which show strong similarities with the two elementary steps of LBM. As regards to data access, Eq. 15 is equivalent to the usual LBM in-place propagation with the additional constraint of retrieving $\rho(\boldsymbol{x}, t)$ and $\boldsymbol{u}(\boldsymbol{x}, t)$ in order to compute

1. **for all** time step t **do**
2. **for all** mesh point $\boldsymbol{x}$ **do**
3. load $\rho(\boldsymbol{x}, t)$ and $\boldsymbol{u}(\boldsymbol{x}, t)$
4. **for all** index α **do**
5. load $f_\alpha^*(\boldsymbol{x} - \boldsymbol{\xi}_\alpha, t)$
6. compute $f_\alpha(\boldsymbol{x}, t+1)$ (Eq. 15)
7. **end for**
8. compute $\rho(\boldsymbol{x}, t+1)$ (Eq. 6)
9. compute $\boldsymbol{u}(\boldsymbol{x}, t+1)$ (Eq. 7)
10. store $\rho(\boldsymbol{x}, t+1)$ and $\boldsymbol{u}(\boldsymbol{x}, t+1)$
11. **for all** index α **do**
12. compute $f_\alpha^*(\boldsymbol{x}, t+1)$ (Eq. 16)
13. store $f_\alpha^*(\boldsymbol{x}, t+1)$
14. **end for**
15. **end for**
16. **end for**

Algorithm 1: General formulation of the two-step LW-ACM.

1. **for all** time step t **do**
2. **for all** mesh point $\boldsymbol{x}$ **do**
3. **for all** index α **do**
4. load $\rho(\boldsymbol{x} - \boldsymbol{\xi}_\alpha, t)$ and $\boldsymbol{u}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha, t)$
5. compute $f_\alpha^{(\mathrm{e})}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha, t)$ (Eq. 8)
6. compute $f_\alpha^{(\mathrm{e,o})}(\boldsymbol{x} - \boldsymbol{\xi}_\alpha, t)$ (Eq. 10)
7. **end for**
8. **for all** index α **do**
9. compute $f_\alpha(\boldsymbol{x}, t+1)$ (Eq. 11)
10. **end for**
11. compute $\rho(\boldsymbol{x}, t+1)$ (Eq. 6)
12. compute $\boldsymbol{u}(\boldsymbol{x}, t+1)$ (Eq. 7)
13. store $\rho(\boldsymbol{x}, t+1)$ and $\boldsymbol{u}(\boldsymbol{x}, t+1)$
14. **end for**
15. **end for**

Algorithm 2: General formulation of the single-step LW-ACM.

$f_\alpha^{(\mathrm{e},\mathrm{o})}(\boldsymbol{x}, t)$. Eq. 16 is equivalent to the LBM collision step insofar as only local information (with respect to both time and space) are required. It is worth noting that, with this formulation, the f_α are disposable variables and that only ρ, $\boldsymbol{u}$, and the f_α^* need to be stored globally between two iterations. The corresponding algorithm is summarised in Algorithm 1.

From an implementation standpoint, the two-step formulation of the LW-ACM is very close to LBM. Thus, the adaptation of an existing LBM code to LW-ACM should be straightforward in general. In a memory-bound context, as for most GPU implementations of the LBM, the additional cost of loading and storing ρ and $\boldsymbol{u}$ at each time step is likely to have a noticeable, although slight, impact on performance. When using the D3Q19 stencil for instance, the amount of data transferred is increased by $4/19 \approx 21\,\%$.

Considering the right-hand side of Eq. 11, one sees that the LW-ACM updating rule is fully expressed in terms of known functions of ρ and $\boldsymbol{u}$. This suggests an alternative implementation approach, outlined by Algorithm 2, in which only ρ and $\boldsymbol{u}$ are kept in memory.

Compared to LBM, the single-step formulation considerably reduces the memory consumption. However, in the three dimensional case, the number of required read operations per time step is multiplied by 4. To be of practical interest in a memory-bound situation, such approach must therefore be coupled with an appropriate strategy to minimise read redundancy.

3. Implementations

3.1. General-purpose computing on GPUs

General-purpose computing on graphics processing units is still an emerging field. A thorough description of the CUDA technology being beyond the scope of this article, we refer the reader to manuals such as the CUDA programming guide [12]. For the sake of clarity, however, we shall summarise some of the specific aspects of GPU programming.

The CUDA programming paradigm is referred to as single instruction multiple threads (SIMT). A CUDA program consists of sequential code run by the host system and at least one function, named *kernel*, which is off-loaded to the computing device at the appropriate time and processed in parallel threads. Each thread is an instance of the kernel with its own local variables. To launch a kernel, it is necessary to specify an *execution grid* which consists of a set of identical *thread blocks*.

The two-level structure of the execution grid is due to architectural constraints. A CUDA enabled GPU is formed by several streaming multiprocessors (SMs) each of one containing a given number of scalar processors (SPs)[1] and a small shared memory. Within an SM, the SPs are strongly coupled: an SP may either process the current instruction or remain idle (which might occur in case of conditional branching). At GPU level, the SMs are not synchronised and may only communicate through the off-chip device memory.

A thread block must fit into one single SM which, on one hand, implies strong limitations on its size and resource consumption, but, on the other hand, enables

[1] Depending on the hardware generation, the number of SPs per SM ranges from 8 to 32.

efficient data transfer and synchronisation between threads. At global level, on the contrary, synchronisation and data transfer operations are likely to have a significant cost and should be avoided as much as possible.

3.2. Basic implementation

Our first attempt to implement the LW-ACM was based on a modified version of the TheLMA[2] framework hereafter referred to as *TheLMA**. Consisting of multiple software components, TheLMA is devoted to the implementation of LBM solvers on GPU based systems [2]. A main computation kernel is responsible for updating the lattice, performing both propagation and collision. This kernel is invoked for each time step, therefore enforcing global synchronisation. Moreover, in order to avoid read-after-write hazards, two instances of the lattice are kept in memory. Although less memory consuming approaches are possible, this method proves to be the most convenient and efficient in practice, allowing to keep the code as simple as possible, even with complex boundary conditions.

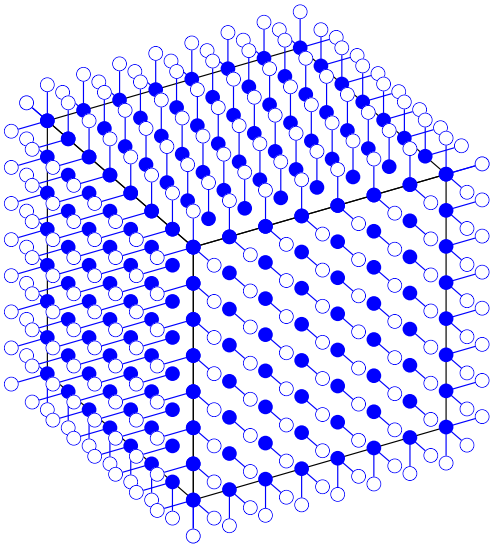
In order to gain advantage from the massive parallelism of GPUs, each node of the computation domain is handled by a specific thread. Since the blocks within the execution grid and the threads within the blocks are referenced by three-dimensional indices, there are numerous ways to set the correspondence between the grid and the computation domain. For simulations in three dimensions, experience shows that the most efficient approach consist in using a two-dimensional grid of one-dimensional blocks. The lattice is stored in a four-dimensional array of floating point numbers, the fastest-varying dimension corresponding to the direction of the blocks. This setup enables the SMs to perform coalesced accesses to device memory resulting in high data transfer rates. Moreover, practice shows that choosing the velocity index as the second fastest-varying dimension has a positive impact on cache reuse and thus on performance.

The TheLMA* version of LW-ACM implements Algorithm 1. Being designed for versatility, very few modifications were required to adapt the TheLMA framework to LW-ACM. Only about 150 lines of code needed to be edited, a first functional version being available after less than 3 days of work. The most notable change concerns the fluid density and velocity which are usually kept in separate arrays since these variables need not be saved at every time step. In the present case, ρ and the components of $\boldsymbol{u}$ are stored as supplementary f_α variables with $\alpha = 19, 20, 21, 22$. This modification furthermore improves cache reuse and preserves simple array index computations in the main kernel.

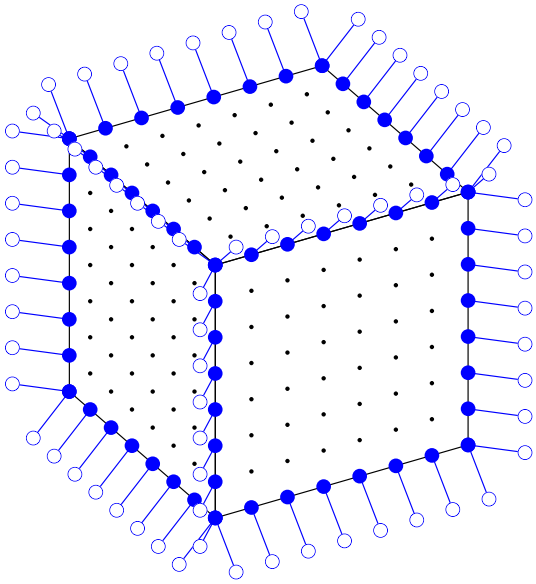
3.3. Optimised implementation

Our second implementation of the LW-ACM is based on Algorithm 2. Although some peripheral components of TheLMA could be recycled, the core elements of this new program, which we named *Louise*, had to be written from scratch. As a matter of fact, the constraints in terms of data transfer are completely different, the key aspect being the read redundancy. To address this issue, we chose to fetch in shared memory the fluid density and velocity associated to each block and its immediate neighbourhood, which we further refer to as its *halo*. As for TheLMA

[2] TheLMA stands for *Thermal LBM on Many-core Architectures*, thermal simulations of the indoor and outdoor environment of buildings being the primary goal of this project.



(a) Faces of the halo.



(b) Edges of the halo.

Figure 2: Data access to the halo of a block for the main kernel of Louise — *The plain discs represent the active threads whereas the hollow ones represent the nodes from which ρ and $\boldsymbol{u}$ are read.*

based solvers, we assign a specific thread to each node, invoke the main computation kernel at each time step, and keep two instances of the fluid data in global memory. However, instead of using four-dimensional arrays to store the fluid data, we use three-dimensional arrays of `float4` structures containing the density and the velocity components. This set-up, which is made possible by the use of a shared buffer at block level, preserves memory access coalescence while improving data locality in global memory.

In order to optimise the number of read operations with respect to the number of threads, the shape of the blocks must be cubic and the size must be as large as possible. Moreover, since coalesced memory accesses are issued by groups of 32 threads named *warps*[3], the number of threads in the blocks should be a multiple of 32. Taking into account the amount of shared memory provided on present CUDA enabled GPUs, we therefore chose to use $8 \times 8 \times 8$ blocks. In the main kernel, each thread starts with loading the fluid density and velocity at its associated node. The threads located at the faces or the edges of the current block are furthermore responsible for loading the fluid data from the corresponding nodes of the halo (see Fig. 2). It should be mentioned that, when using the D3Q19 stencil as in our case, the data from the vertices of the halo are not required and thus are not loaded. Once all data is available, the threads update ρ and $\boldsymbol{u}$ for their associated node and write these values to global memory.

Using the proposed approach, there are 992 read operations[4] and 512 write operations to global memory per block and per time step. Each block being assigned to 512 nodes, the read redundancy ratio is therefore less than 2. When comparing Louise to a TheLMA based LBM solver, the amount of data read is reduced by a factor of 2.44, the amount of data written is reduced by a factor of 4.75, and the global memory consumption is reduced by a factor of 5.25.

4. Performance study

4.1. Methodology

In order to demonstrate the relevance of our optimisations, we studied performance of both the TheLMA* and the Louise programs as well as of a TheLMA based D3Q19 LBM solver. We chose to simulate the well-know lid-driven cubic cavity in single precision. This test case consists of a closed cavity with five solid walls and a top lid on which a constant velocity is imposed to the fluid. Both the TheLMA and the TheLMA* solvers implement the simple bounce-back (SBB) boundary condition for the walls. Following [9], we impose the top lid velocity $\boldsymbol{u}_0$ by adding :

$$\frac{2}{c_s^2} w_\alpha \rho \boldsymbol{\xi}_\alpha \cdot \boldsymbol{u}_0 \tag{17}$$

to the f_α variables after bounce-back. As regards to Louise, since LW-ACM can accommodate with either finite difference or LBM techniques, and since retrieving the values of the f_α at the former time step would lead to additional memory accesses, we chose to implement usual finite difference boundary conditions. While

[3] The size of a warp is actually hardware-dependent and might be different with future CUDA GPUs.

[4] This evaluation is only valid in the bulk of the computation domain, since the application of boundary conditions might alter the data access pattern.

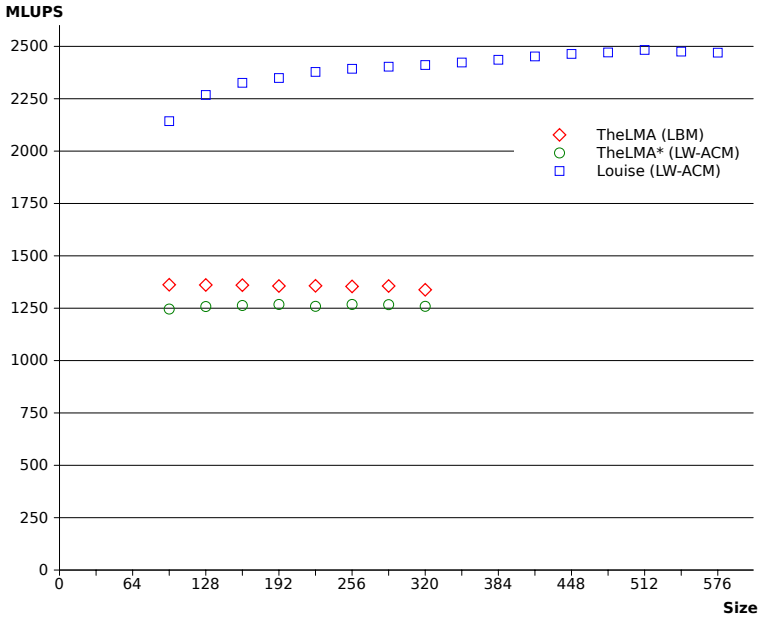


Figure 3: Performance comparison on the lid-driven cubic cavity — *Performance is reported in million lattice node updates per second (MLUPS).*

Size	TheLMA	TheLMA*	Louise
96	89.6	99.2	43.6
128	89.6	100.2	46.1
160	89.5	100.6	47.3
192	89.2	101.0	47.8
224	89.3	100.3	48.4
256	89.1	101.0	48.7
288	89.2	100.9	48.9
320	88.0	100.3	49.1
352	—	—	49.3
384	—	—	49.6
416	—	—	49.9
448	—	—	50.1
480	—	—	50.3
512	—	—	50.5
544	—	—	50.4
576	—	—	50.3

Table 1: Ratio (in %) of the data throughput to the maximum sustained throughput for the lid-driven cubic cavity test case.

preserving performance, this choice enabled us to compare the effects of both types of boundary conditions.

We performed the simulations on a GeForce GTX Titan commodity graphics card featuring a Kepler GK110 GPU and 6 GB of device memory. According to the `bandwidthTest` program shipped with the CUDA software development kit, the maximum sustained data throughput between GPU and device memory is approximatively 231 GB/s, which is almost twice as much as for the Fermi, i.e. the former CUDA GPU generation. The graphics card was hosted on a Debian GNU/Linux 6.0 workstation running version 319.23 of the Nvidia device driver. Preliminary tests showed that the device driver enforces an aggressive power management policy, downscaling the processor frequency to keep the core temperature within optimal range. Recorded performance may thus considerably vary, depending on the start temperature and the duration of the run. To get reproducible results and valid comparisons, it is therefore necessary to ensure identical start temperatures and run times. In the present study, we chose to perform short-time runs (about 30 s) at low start temperature (42 °C), hence staying at the maximum frequency level. In practice, for long-time computations, experience shows that the obtained performance is about 15 % less than the values reported hereafter.

4.2. Results and discussion

Performance is reported in MLUPS, i.e. million lattice node updates per second, which is the usual performance metric for LBM. The size of the cavity ranges from 96 to the largest possible, i.e. 576 for Louise and 320 for TheLMA and TheLMA*. The results are plotted in Fig. 3. In addition, the ratio of the data throughput estimated from performance to the maximum sustained throughput is given in Tab. 1.

Inspecting Tab. 1 leads to the conclusion that the performance of TheLMA and TheLMA* is memory-bound, the former outperforming the later by about 7 % on average. There appears to be significant cache reuse[5], the ratios being close to or even greater than 100 %. The difference in performance between the TheLMA and TheLMA* versions is less than expected when considering the amount of data to be transferred at each time step. However, opposite to the loading and storing of the f_α variables, the additional memory access operations required by TheLMA* are all well-aligned and thus more efficient.

The Louise program outperforms the TheLMA based solver by a factor of up to 1.8, however the performance increase is not proportional to the gains in data transfer. Using the CUDA profiler shows that, because of the synchronisation barrier required after gathering the data at the beginning of the main computation kernel, a large proportion of the load operations are stalled, thus exposing the high latency of the device memory. It is also worth mentioning that the performance of Louise grows with the size of the cavity, which is likely to be caused by an increasing cache reuse. Indeed, since the blocks are dispatched to the SMs in an ordered fashion, a larger computation domain increases the probability to have consecutive blocks processed concurrently, thus improving the data locality. With Louise, performance goes up to nearly 2500 MLUPS which is close to the performance re-

[5] When a SM starts processing a block of threads, 19 coalesced read operations are issued and the corresponding memory segments are loaded from device memory through L2 cache. Because of the address shift induced by propagation, 10 out of the 19 transactions are not aligned and some of the loaded segments can be reused for neighbouring blocks.

ported for a TheLMA based multi-GPU solver running on a cluster of eight Fermi GPUs [14].

5. Validation study

5.1. Reference data

Although the LW-ACM was carefully validated in [3], no high-resolution validation study of the LW-ACM has been published so far to the best of our knowledge. For this purpose, we generated highly accurate reference data of the lid-driven cubic cavity at Reynolds number $\mathrm{Re} = 2000$, using a spectral and mortar element analysis program developed using the OpenSPECULOOS toolbox [1, 5]. We performed the simulation on 8^3 elements with Gauss–Lobato–Legendre polynomials of degree 12 and a dimensionless time step duration of 10^{-3}. We obtained a dimensionless time-to-solution of 18.4 using the convergence criterion:

$$\|\boldsymbol{U}_{n+1} - \boldsymbol{U}_n\|_2 < 10^{-2} \tag{18}$$

where $\|\cdot\|_2$ is the ℓ^2-norm and $\boldsymbol{U}_n$ is the dimensionless velocity field at time step n. The simulation ran for 64 hours on 256 CPU cores of a cluster featuring double socket Xeon E5-2670 nodes and InfiniBand FDR interconnect.

5.2. Results and discussion

In order to evaluate the accuracy of LW-ACM and to make comparison with LBM, we simulated the lid-driven cubic cavity in single precision at $\mathrm{Re} = 2000$ with both Louise and TheLMA* (since the implemented boundary conditions are different) as well as with the TheLMA version which uses a multiple-relaxation-time (MRT) lattice Boltzmann model [7]. We performed the simulation for a dimensionless duration of 18.4, as for our reference data, and used the same range of resolutions as in our performance study.

Comparison between the LW-ACM and LBM simulations and our reference is made by subtracting the resulting normalised velocity field to the values interpolated from the reference data on the appropriate Cartesian grid. The ℓ^2-norm of the discrepancy fields are plotted in Fig. 4, whereas a detailed view of these discrepancies in a cavity of size 128 is provided by Figs. 5, 6, and 7.

Fig. 4 shows that both LW-ACM and LBM converge towards the reference solution at approximatively the same rate. Moreover, LW-ACM appears to be almost as accurate as LBM, except for the coarser resolutions in the case of TheLMA*. It is worth mentioning that using the Louise program with a Kepler based accelerator, a discrepancy norm of less than 1 % is achieved for a cavity of size 224 within less than 3 minutes of computations.

Considering Figs. 5, 6, and 7, we see that the discrepancies are mainly located near the top lid. For LBM, the most important discrepancies are close to the top edge of the downstream wall, where the pressure gradients are the higher. In the case of TheLMA*, the discrepancies are less severe but more extended than with the two other programs, resulting in a larger ℓ^2-norm. With Louise, the major discrepancies are along the stream-wise edges of the top lid. Improved boundary conditions on the edges could possibly cure this defect.

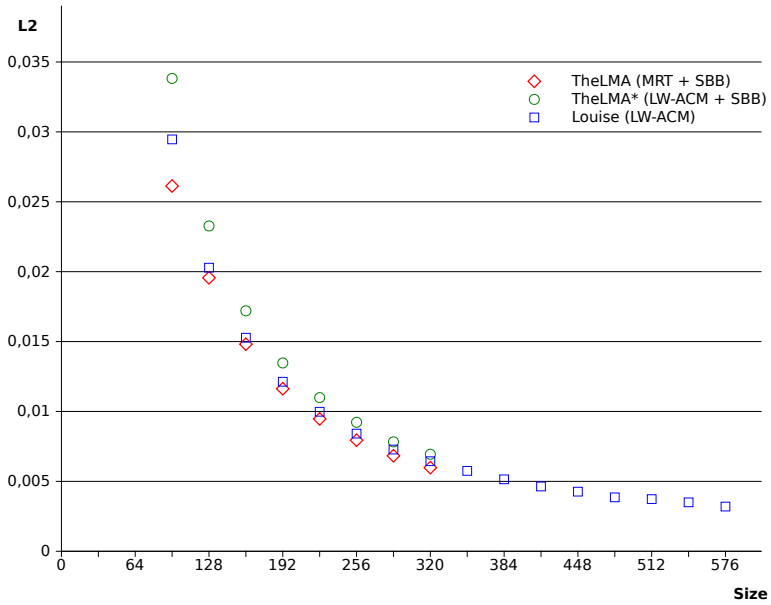


Figure 4: ℓ^2-norm of the velocity discrepancy with respect to reference data for TheLMA, TheLMA* and Louise on the lid-driven cubic cavity at $\mathrm{Re} = 2000$.

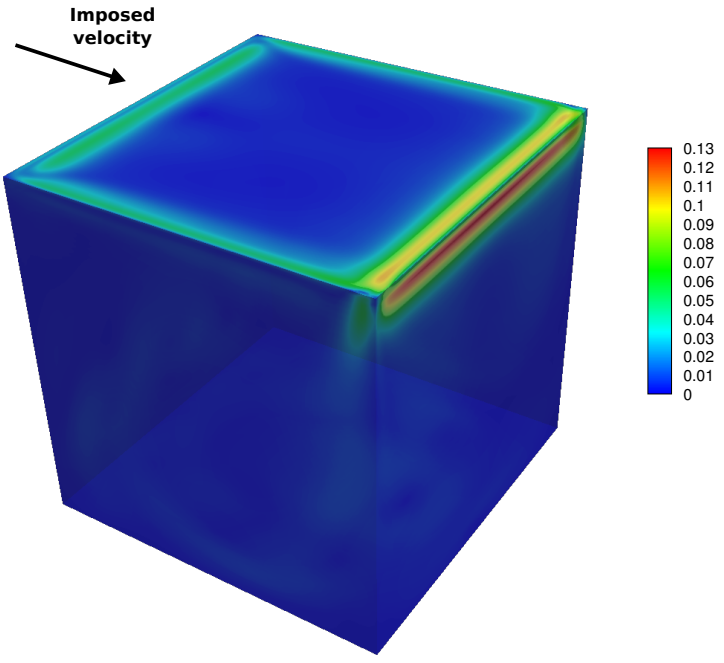


Figure 5: Velocity discrepancy with respect to reference data for TheLMA.

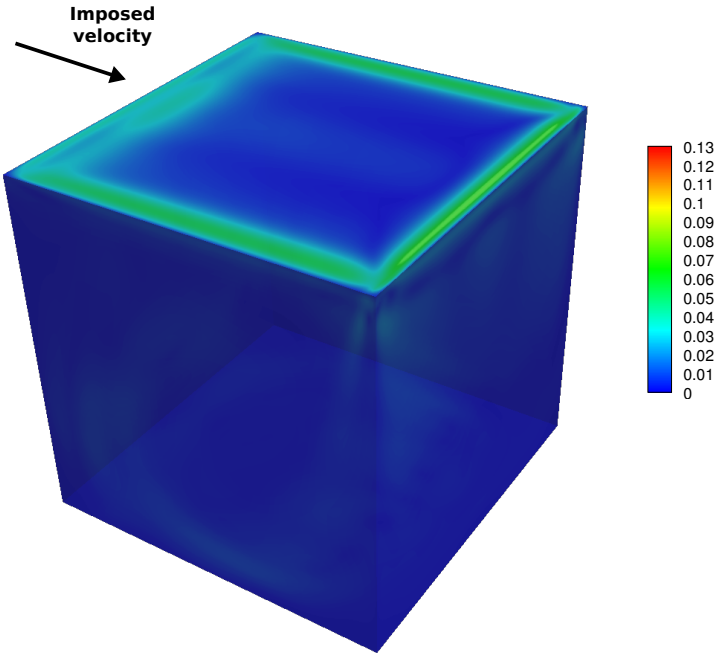


Figure 6: Velocity discrepancy with respect to reference data for TheLMA*.

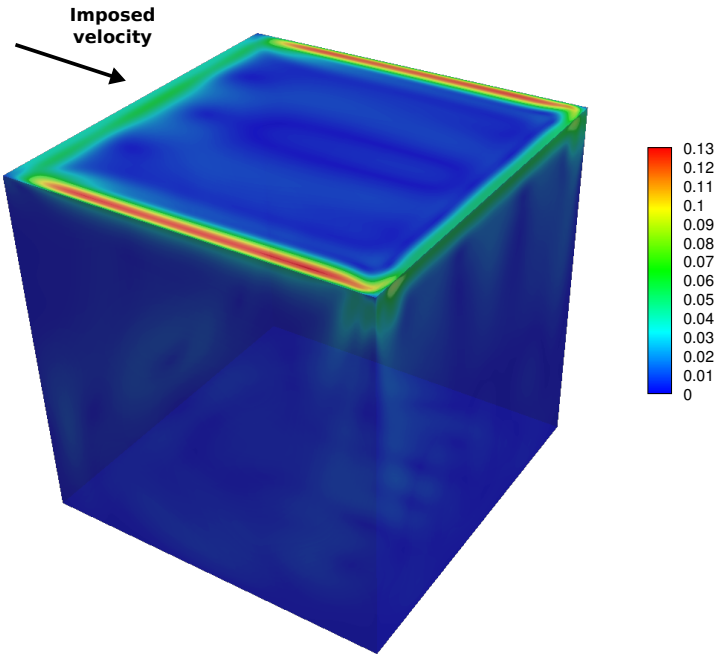


Figure 7: Velocity discrepancy with respect to reference data for Louise.

6. Conclusions

In the present work, we describe two implementations of the LW-ACM on a CUDA enabled GPU. The approach chosen for the Louise program appears very promising. Compared to LBM, performance with the latest CUDA generation is increased by a factor of up to 1.8. The global memory requirements are reduced by a factor of 5.25, making possible to handle up to 201 million nodes instead of 38 using a 6 GB computing device. The TheLMA* version is of less practical interest. However, it proves that an existing GPU LBM might easily be adapted to LW-ACM with only slight performance loss.

The large scale validation enabled by these high-performance implementations shows that LW-ACM is able to reach excellent accuracy, at least at moderate Reynolds number, in a very short amount of time. Implementations of the LW-ACM on massively parallel processors could become instruments of choice for CFD simulations in engineering applications.

References

[1] SPECtral Unstructured eLements Object-Oriented System (SPECULOOS). www.sourceforge.net/projects/openspeculoos.

[2] Thermal LBM on Many-core Architectures (TheLMA). www.thelma-project.info.

[3] P. Asinari, T. Ohwada, E. Chiavazzo, and A.F. Di Rienzo. Link-wise Artificial Compressibility Method. *Journal of Computational Physics*, 231(15):5109–5143, 2012.

[4] P. L. Bhatnagar, E. P. Gross, and M. Krook. A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems. *Physical Review*, 94(3):511–525, 1954.

[5] C. Bosshard, R. Bouffanais, C. Clémençon, M. Deville, N. Fiétier, R. Gruber, S. Kehtari, V. Keller, and J. Latt. Computational Performance of a Parallelized Three-Dimensional High-Order Spectral Element Toolbox. In *Advanced Parallel Processing Technologies*, volume 5737 of *Lecture Notes in Computer Science*, pages 323–329. Springer, 2009.

[6] A. J. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of computational physics*, 2(1):12–26, 1967.

[7] D. d'Humières, I. Ginzburg, M. Krafczyk, P. Lallemand, and L.S. Luo. Multiple-relaxation-time lattice Boltzmann models in three dimensions. *Philosophical Transactions of the Royal Society A*, 360:437–451, 2002.

[8] S Geller, S Uphoff, and M Krafczyk. Turbulent jet computations based on mrt and cascaded lattice boltzmann models. *Computers & Mathematics with Applications*, 65(12):1956–1966, 2013.

[9] A. J. C. Ladd and R. Verberg. Lattice-boltzmann simulations of particle-fluid suspensions. *Journal of Statistical Physics*, 104(5-6):1191–1251, 2001.

[10] X.L. Luo, Z.L. Gu, K.B. Lei, S. Wang, and K. Kase. A three-dimensional Cartesian cut cell method for incompressible viscous flow with irregular domains. *International Journal for Numerical Methods in Fluids*, 69(12):1939–1959, 2012.

[11] R. Mittal and G. Iaccarino. Immersed boundary methods. *Annual Reviews of Fluid Mechanics*, 37:239–261, 2005.

[12] NVIDIA. *Compute Unified Device Architecture Programming Guide version 5.5*, July 2013.

[13] C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux. A New Approach to the Lattice Boltzmann Method for Graphics Processing Units. *Computers and Mathematics with Applications*, 12(61):3628–3638, 2011.

[14] C. Obrecht, F. Kuznik, B. Tourancheau, and J.-J. Roux. Scalable lattice Boltzmann solvers for CUDA GPU clusters. *Parallel Computing*, 39(6-7):259–270, 2013.

[15] R. Témam. *Navier–Stokes equations: Theory and numerical analysis*. North-Holland, 1979.

[16] J. Tölke and M. Krafczyk. TeraFLOP computing on a desktop PC with GPUs for 3D CFD. *International Journal of Computational Fluid Dynamics*, 22(7): 443–456, 2008.